Parallel Image Compositing API

Byron Alcorn & Randall Frank
HP Workstations & Lawrence Livermore National Labs

Workshop on Commodity-Based Visualization Clusters
“Standard” commodity clusters
- Nodes + GPU + Interconnect
- PowerWalls
- Remote displays

Multiple goals
- Interaction/VR: High frame rates
- PowerWalls: Large pixel counts
- Data scaling: High polygon/fill rates
- Image Quality: Full scene anti-aliasing

Hardware for “compositing” has focused on:
- Application transparency
- Parallel rendering models

Compositing solutions vary considerably:
- sv6, Sepia, SGE, Lightning2
- Many software systems
An Idealized Visualization Environment

Workshop on Commodity-Based Visualization Clusters
Inhibitors to Compositor Adoption

- Small market
  - Few applications
  - No such thing as a “standard” cluster
  - No common rendering infrastructure for parallel applications
- No common API for compositors
  - Application transparent modes are scalability limited
  - Invasive/custom interfaces to devices for “special features”
- Lack of 2D integration with 3D
- Much of the 3D intensive SW was developed for SMP machines with limited graphics performance
  - Scene graph management
  - Preparing data for the rendering pipeline
- Hardware limitations exist
  - Capabilities of COTS graphics cards
  - Bandwidth available for image fragments
Parallel Image Compositing API (PICA)

- System developers and early adopting app developers
  - Mostly HP, PNNL and LLNL employees
  - Informal API discussions
- Goals
  - Abstraction for distributed image composition
  - Provide an open source API adoptable by ISVs
  - Provides a platform for SW implementations
  - Target major hardware compositors
Toolkits: “Scene Graphs”, primitive generation
- OpenRM
- VTK
- OpenSG / Open Scene Graph

Chromium: parallel OpenGL API

DMX: distributed X11 / windowing

Merlot: remote image transport interfaces

PICA: “compositor” abstraction
Underling Assumptions

- API targets parallel applications
  - Simple parallel model assumptions
  - Application can pass messages to itself
- API must handle various input sources
  - Region of graphics card memory
  - Software rendering to main memory
  - Must support “windowed” applications
- API must provide a complete compositing abstraction
  - Abstract the concept of composite “ordering”
  - Compositing functions covering common usage
- Independent of graphics API
- Must abstract all current compositor forms
  - Multiple compositors available in the same cluster
  - DVI based, network based, software, etc
Basic PICA Abstractions

- Application nodes
  - Source of image fragments (ifrag), includes rendering resource
  - Application running on every node

- Application generates a sequence of “Frames”
  - Frames are sequenced by IDs
  - Limited queries Supported via frame IDs
  - Multiple frame “channels” for stereo

- Frames are built from multiple ifrags
  - An ifrag is a rectangle of augmented (e.g. $\alpha$ & depth) pixels
  - Individual nodes can submit multiple ifrags
  - ifrags can be located anywhere in a frame
  - ifrags are tagged with an “order” number within each frame

- “OpenGL”-style compositing pipeline
  - Multiple conceptual “stages” of compositing supported
  - The order of ifrag introduction can be application specified
Compositing Operation in PICA

Cluster interconnect

Nodes
- Workstation 0
  - Node 0
  - Node 1
- Workstation 1
  - Node 2
  - Node 3
- Workstation 2
  - Node 4

iffrags
- Generate imagery

gfx

Context
- Ordered image fragments
- Realization of a compositor

Compositor

Frame
- Output complete images

PICA API wrapper

TeraScale Browser
- Windowed application, PowerWall, other devices

Workshop on Commodity-Based Visualization Clusters
Workshop on Commodity-Based Visualization Clusters

Compositing Operation in PICA: Details

Cluster interconnected

Nodes

Workstation 0
  Node 0
gfx
  Node 1
gfx

Workstation 1
  Node 2
gfx
  Node 3
gfx

Workstation 2
  Node 4
gfx

PICA API wrapper

Order #s

0

Nodes

iffrags

Generate imagery

Ordered image fragments

TeraScale Browser

“Conceptual” Compositor

Application level abstraction, physical layers are free to implement as desired

Alpha-blending + Z-buffering
Order preserving
Accepts “orders” 0-3

Z-buffering
Non-order preserving
Accepts “order” 4

Compositor Stages

Each realizes a “composite” : Inputs 2 iffrags, outputs an ifrag

Frame

Scale/Bias → α-testing → Z-buffer → Blending → Logic Ops → Masking

Output complete images

Workshop on Commodity-Based Visualization Clusters
Basic PICA Operation

- Compositing “context” negotiated (one node)
  - Includes compositing pipeline definition
  - Hardware is allocated
- Context ID is passed to nodes by application
- Context is “bound” locally (all nodes) to realize the system
- Application starts a frame (all nodes)
  - Application renders graphics (generates ifrags)
  - ifrags are passed to local context
- Application ends the frame (all nodes)
  - Composite may occur asynchronously
- Basic query functions allow for application feedback
Technical Details

- Compositing happens at the application rate, not at the display rate
- Designed to support 99% of applications
  - Sorted, alpha blending (e.g. volume rendering)
  - Tiling
  - “Overlays” (e.g. annotations, heads-up-displays)
- Advanced composites supported as well
  - Anti-aliasing
  - HW assisted “transparency”
- Provides mechanisms for application “hinting”
  - Performance optimizations (e.g. BSP composite trees)
  - “Specialized” features (e.g. incomplete crossbars)
- Compositing API mostly independent of graphics API
  - Some calls restricted (glXSwapBuffer, glXCreateContext)
  - Special “window manager” specific create context calls
Current Status

- First revision of the specification is complete
  - Written as a communication tool
  - “Human readable” specification next
- Development efforts
  - “C” Stubs written for the API
  - Compiles both apps and compositor
  - Tiered shared library dispatch done
  - Simple software compositor under development
  - Test application under development
- Continued investigations/discussions
  - Mapping to various hardware systems
  - Application transparency issues
  - Parallel system model and security issues
Future Steps

- Work out additional high level technical details
  - Opportunities for performance optimizations
  - Node allocation issues (e.g. multiple graphics pipelines in a node)
  - How to choose a compositor
- Work out next level of details
  - Sample application
  - Partial compositor
- Create sample implementation
- Write drivers to support a HW compositor
- Get API feedback from a wider audience
  - Researchers
  - Application writers
  - HW providers
The LLNL work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Workshop on Commodity-Based Visualization Clusters